7.Vorlesung Grundlagen der Informatik

Dr. Christian Baun

Hochschule Darmstadt
Fachbereich Informatik
christian.baun@h-da.de

24.11.2011

Wiederholung vom letzten Mal

- Zeichen-/Blockorientierte Geräte
- Daten von Ein- und Ausgabegeräten lesen
- Adressraum
- Speicheradressierung und Speicherverwaltung
- Speicherpartitionierung
  - Statische Partitionierung
  - Dynamische Partitionierung
  - Buddy-Verfahren
- Virtueller Speicher
  - Memory Management Unit
  - Seitenorientierter Speicher (Paging)
  - Segmentorientierter Speicher (Segmentierung)

Heute

- Klassifikationen von Betriebssystemen
  - Betriebsarten (Stapelbetrieb, Dialogbetrieb)
  - Singletasking und Multitasking
  - Einzelbenutzerbetrieb und Mehrbenutzerbetrieb
  - Ein-Prozessor- und Mehr-Prozessor-Betriebssysteme
  - Echtzeitbetriebssysteme
- Prozesskontext
  - Benutzerkontext
  - Hardwarekontext
  - Systemkontext
- Prozessmodelle
- Prozesstabellen und Prozesskontrollblöcke
- Zustandslisten

Klassifikationen von Betriebssystemen

- Die existierenden Betriebssysteme können nach unterschiedlichen Kriterien klassifiziert werden
  - **Einzel-/Mehrprogrammbetrieb**
  - **Ein-/Mehr-Prozessor-Fähigkeit**
  - **Echtzeitbetrieb**
  - **Kernelarchitektur**

Einzelprogrammbetrieb und Mehrprogrammbetrieb

- **Einzelprogrammbetrieb** (Singletasking)
  - Zu jedem Zeitpunkt läuft nur ein einziges Programm
  - Mehrere gestartete Programme werden nacheinander ausgeführt
- **Mehrprogrammbetrieb** (Multitasking)
  - Mehrere Programme können gleichzeitig (bei mehreren CPUs) oder zeitlich verschachtelt (quasi-parallel) ausgeführt werden
  - Die CPU kann nicht nur am Ende eines Programms einem anderen Programm zugeteilt werden, sondern auch zwischen den Programmen wechseln
    $\Longrightarrow$ **Timesharing**: Mehrere Benutzer arbeiten gleichzeitig am System

Task, Prozess, Aufgabe, Auftrag,...

Der Begriff **Task** ist gleichzusetzen mit dem Begriff **Prozess** oder aus Anwendersicht **Aufgabe** bzw. **Auftrag**

Warum Mehrprogrammbetrieb (Multitasking)?

- Bei **Mehrprogrammbetrieb** laufen mehrere Prozesse nebenläufig
- Die Prozesse werden in kurzen Abständen, abwechselnd aktiviert
  - Dadurch entsteht der Eindruck der Gleichzeitigkeit
- Beim Umschalten von einem Prozess zu anderen, entsteht ein Overhead
  - Dennoch bringt Mehrprogrammbetrieb immer Vorteile mit sich
- Prozesse müssen häufig auf äußere Ereignisse warten
  - Dabei kann es sich um Benutzereingaben, Eingabe/Ausgabe-Operationen von Peripheriegeräten oder einfach das Warten auf eine Nachricht eines anderen Programms handeln
- Durch Mehrprogrammbetrieb können Prozesse, die auf ankommende E-Mails, erfolgreiche Datenbankoperationen, geschriebene Daten auf der Festplatte oder ähnliches warten, in den Hintergrund geschickt werden
  - Andere Prozesse kommen so früher zum Einsatz
- Der Overhead, der bei der quasiparallelen Abarbeitung von Programmen durch die Programmwechsel entsteht, ist im Vergleich zum Geschwindigkeitszuwachs zu vernachlässigen

Einzelbenutzerbetrieb und Mehrbenutzerbetrieb (1)

- Betriebssysteme lassen sich auch nach der **Anzahl der gleichzeitig am Computer arbeitenden Benutzer** klassifizieren
- **Einzelbenutzerbetrieb** (Single-User)
  - Der Computer steht immer nur einem einzigen Benutzer zur Verfügung
  - Es gibt Single-User-Betriebssysteme mit Single- und mit Multitasking
  - Bekannte Beispiele: MS-DOS, Windows 3x/95/98, OS/2, BeOS
- **Mehrbenutzerbetrieb** (Multi-User)
  - Mehrere Benutzer können gleichzeitig mit dem Computer arbeiten
  - Benutzer teilen sich die Systemleistung
  - Systemressourcen müssen möglichst gerecht verteilt werden
  - Benutzer müssen (durch Passwörter) identifiziert werden
  - Zugriffe auf Daten/Prozesse anderer Benutzer werden verhindert
  - Bekannte Beispiele: Linux/UNIX, MacOS X,...

Einzelbenutzerbetrieb und Mehrbenutzerbetrieb (2)

- Die Desktop/Workstation-Versionen von Windows NT/XP/Vista sind nur **halbe Multi-User-Betriebssysteme**
  - Verschiedene Benutzer können nur nacheinander am System arbeiten, aber die Daten und Prozesse der Benutzer sind voreinander geschützt

	Single-User	**Multi-User**
Singletasking	MS-DOS, Palm OS	—
Multitasking	OS/2, Windows 3x/95/98, BeOS, MacOS 8x/9x, AmigaOS	Linux/UNIX, MacOS X Windows NT/2000 Terminal Server

Betriebssysteme für einen oder mehr Prozessoren

- **Ein-Prozessor-Betriebssysteme**
  - Für Computer mit einem Hauptprozessor (CPU)
  - Singletasking-Betriebssysteme sind üblicherweise auch Ein-Prozessor-Betriebssysteme
  - Beispiel: MS-DOS für Intel 80x86
- **Mehr-Prozessor-Betriebssysteme**
  - Für Computer mit mehr als nur einem Hauptprozessor
  - Jedem gestarteten Prozess wird ein Hauptprozessor zugeteilt
  - Laufen mehr Prozesse, als Hauptprozessoren existieren, werden die Prozesse zeitlich verschachtelt (quasi-parallel) ausgeführt
  - Mehr-Prozessor-Betriebssysteme sind immer auch Multitasking-Betriebssysteme
  - Läuft ein Betriebssystem auf mehreren CPUs (oder Systemen) verteilt, spricht man von einem **verteilten Betriebssystem**

Heute finden sich mehrere Prozessoren fast ausschließlich in Servern. Multicore ist aber in Servern, Workstations, PCs und Laptops Standard

Echtzeitbetriebssysteme

- Aktuelle Desktop-Betriebssysteme können weiches Echtzeitverhalten für Prozesse mit hoher Priorität garantieren
  - Wegen des unberechenbaren Zeitverhaltens durch Swapping, Hardwareinterrupts etc. kann aber kein hartes Echtzeitverhalten garantiert werden
- Echtzeitbetriebssysteme (*Real-Time Operating Systems*) sind Multitasking-Betriebssysteme mit zusätzlichen Echtzeit-Funktionen für die Einhaltung von Zeitbedingungen und die Vorhersagbarkeit des Prozessverhaltens
- Beim Echtzeitbetrieb sind alle Prozesse ständig betriebsbereit und Ergebnisse stehen in einer vorgegebenen Zeitspanne zur Verfügung
- Wesentliches Kriterium von Echtzeitbetriebssystemen: **Reaktionszeit**
- Unterschiedliche Prioritäten werden berücksichtigt, damit die wichtigsten Prozesse innerhalb gewisser Zeitschranken ausgeführt werden

Harte und Weiche Echtzeitsysteme

- Typische Einsatzgebiete von Echtzeitbetriebssystemen
  - Mobiltelefone
  - Industrielle Kontrollsysteme
  - Roboter
- Beispiele für Echtzeitbetriebssysteme
  - QNX
  - VxWorks
  - LynxOS
  - Nucleus
  - Symbian
  - Windows CE
- Zwei Arten von Echtzeitbetriebssystemen werden unterschieden
  - **Harte Echtzeitsysteme**
  - **Weiche Echtzeitsysteme**

Harte und Weiche Echtzeitsysteme

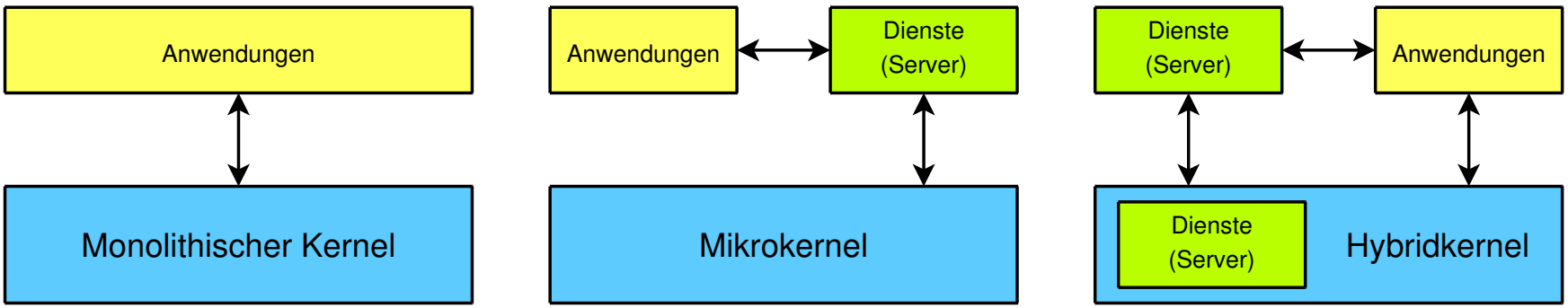
- **Harte Echtzeitsysteme**
  - Zeitvorgaben müssen unbedingt eingehalten werden
  - Verzögerungen können unter keinen Umständen akzeptiert werden
  - Verzögerungen führen zu katastrophalen Folgen und hohen Kosten
  - Ergebnisse sind nutzlos wenn sie zu spät erfolgten
  - Einsatzbeispiele: Schweißroboter, Reaktorsteuerung, ABS, Flugzeugsteuerung, Überwachungssysteme auf der Intensivstation
- **Weiche Echtzeitsysteme**
  - Gewisse Toleranzen sind erlaubt
  - Verzögerungen führen zu geringen, akzeptablen Kosten
  - Einsatzbeispiele: Telefonanlage, Parkschein- oder Fahrkartenautomat, Multimedia-Anwendungen wie Audio/Video on Demand

Architekturen von Echtzeitsystemen

- **Micro-Kernel**
  - Betriebssystemkern selbst läuft als Prozess mit niedrigster Priorität
  - Echtzeit-Kernel (Micro-Kernel) übernimmt das Scheduling
  - Echtzeit-Prozesse besitzen die höchste Priorität
    $\Longrightarrow$ minimale Latenzzeiten
- **Nano-Kernel**
  - Vergleichbar mit dem Micro-Kernel-Ansatz
  - Unterschied: Neben dem Echtzeit-Kernel kann eine beliebige Anzahl anderer Betriebssystemkerne laufen
- **Pico-Kernel, Femto-Kernel, Atto-Kernel,...**
  - Marketingbegriffe der Hersteller von Echtzeitsystemen, um die Winzigkeit ihrer Echtzeit-Kernel hervorzuheben

Betriebssystemaufbau (Kernelarchitekturen)

- Ein entscheidendes Kriterium für den Aufbau eines Betriebssystems ist die Architektur des eingesetzten Kernel (Kern)
  - Die Architekturen unterscheiden sich darin, ob die Funktionen, die sie dem Benutzer und seinen Anwendungen anbieten, **im Kernel** enthalten sind, oder sich **außerhalb des Kernels** als Dienste (Server) befinden
    - Funktionen im Kernel, haben vollen Hardwarezugriff (**Kernelmodus**)
    - Funktionen außerhalb des Kernels können nur ihren virtuellen Speicher zugreifen (**Benutzermodus**)
- Unterschieden werden:
  - **Monolithischer Kernel**
  - **Minimaler Kern** (Mikrokernel)
  - **Hybridkernel** (Makrokernel)



Definition: Prozesse

- Ein **Prozess** ist ein Programm, dass sich in Ausführung bzw. Bearbeitung befindet (lat. procedere = voranschreiten)
- Prozesse sind dynamische Objekte und repräsentieren sequentielle Aktivitäten in einem Computersystem
- Ein Prozess umfasst außer dem Programmcode den **Prozesskontext** und seit UNIX einen geschützten **Prozessadressraum**:
  - Der **Prozesskontext** wird wesentlich durch den Inhalt der Register in der CPU und die Daten im Hauptspeicher bestimmt
  - Der **Prozessadressraum** ist eine Liste von Speicherstellen (virtueller Speicher), in denen der Prozess lesen und schreiben darf
- Auf einem Computersystem sind immer mehrere Aktivitäten (Prozesse) in Ausführung
  - Der Prozessor wird also im raschen Wechsel zwischen den Aktivitäten hin- und hergeschaltet

Der Prozesskontext

- 3 Arten von Kontextinformationen speichert das Betriebssystem:
  - **Benutzerkontext**
    - Daten des Prozesses im zugewiesenen Adressraum (virtuellen Speicher)
  - **Hardwarekontext**
    - Register in der CPU und Seitentabelle
  - **Systemkontext**
    - Informationen, die das Betriebssystem über einen Prozess speichert
- Die Informationen im Hardwarekontext und Systemkontext werden vom Betriebssystem im **Prozesskontrollblock** verwaltet

Hardwarekontext

- Der **Hardwarekontext** umfasst die Inhalte der Register in der CPU zum Zeitpunkt der Prozess-Ausführung und die Seitentabelle
- Register, deren Inhalt bei einem Kontextwechsel gesichert werden muss:
  - Befehlszähler (*Program Counter*)
  - Stackpointer – zeigt auf das oberste Element des Stacks
  - Basepointer – kann auf einen Platz im Stack zeigen
  - Basisregister – zur Adressierung der Anfangsadresse einer Datenstruktur
  - Grenzregister – zur Adressierung der Endadresse einer Datenstruktur
  - Akkumulator – enthält Ergebnisse der Recheneinheit (ALU)
- Diese Informationen sind wichtig, wenn ein Prozess im Rahmen des Multitaskings bei einem Kontextwechsel durch einen anderen Prozess unterbrochen wird

Systemkontext

- Der **Systemkontext** sind die Informationen, die das Betriebssystem über einen Prozess speichert
- Beispiele sind:
  - Eintrag in der Prozesstabelle
  - Prozessnummer (PID)
  - Prozesszustand
  - Information über Eltern- oder Kindprozesse
  - Prioritäten
  - Identifier – Zugriffsrechte auf Ressourcen
  - Quotas – Zur Verfügung stehende Menge der einzelnen Ressourcen
  - Laufzeit
  - Geöffnete Dateien
  - Zugeordnete Geräte

Prozesse

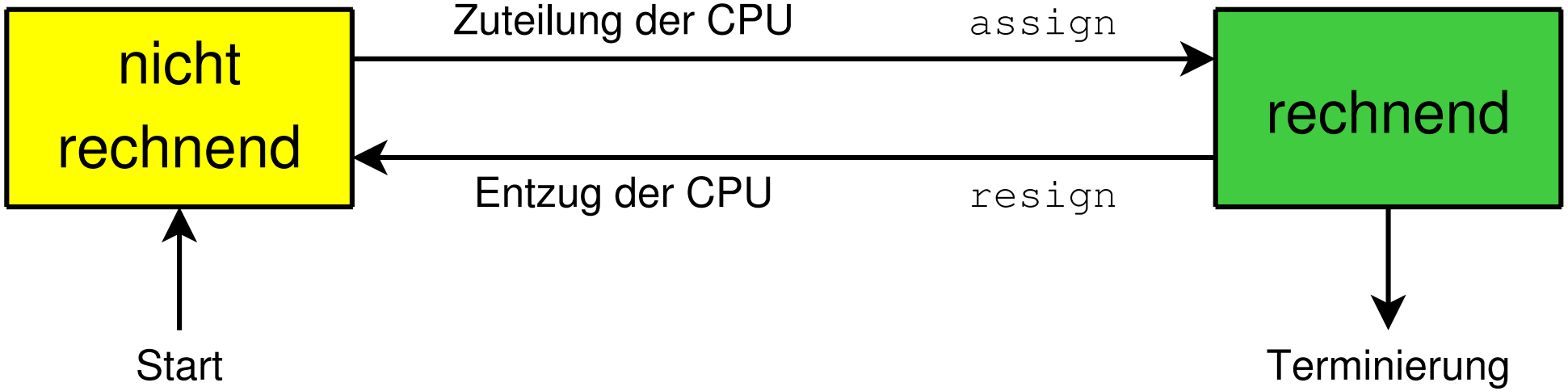
- Jeder Prozess hat seinen eigenen Prozesskontext, der von den Kontexten der anderen Prozesse meist unabhängig ist
- Gibt ein Prozess den Prozessor ab, wird ihr Kontext, also der Inhalt der CPU-Register an einer sicheren Stelle gerettet (zwischengespeichert)
  - Erhält der Prozess wieder den Zugriff auf die CPU, wird der Inhalt des Kontext wiederhergestellt und die Register werden mit den zuvor gespeicherten Daten geladen
- Prozessmanagement und Prozessinteraktion machen den nicht-monopolisierten, geschützten Zugriff auf CPU und Speicher erst möglich
- Jeder Prozess befindet sich zu jedem Zeitpunkt in einem bestimmten **Zustand**
  - Der Zustand gibt an, ob der Prozess gerade auf der CPU ausgeführt wird bzw. ausgeführt werden kann
    $\Longrightarrow$ Zustandsdiagramm der Prozesse

Prozesszustände

- Ein Prozess wird zu Beginn der Programmausführung erzeugt und bei der Terminierung des Programms beendet
- Die Ausführung eines Prozesses kann zur Erzeugung bzw. Beendigung weiter Prozesse führen
- Jeder Prozess befindet sich zu jedem Zeitpunkt in einem Zustand
- Wie viele unterschiedliche Zustände es gibt, hängt vom Prozessmodell des Betriebssystems ab

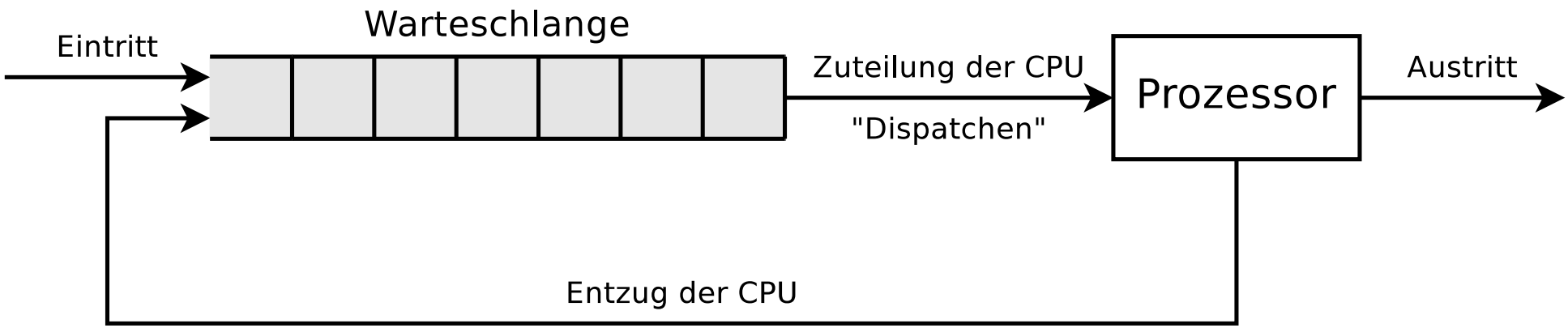
Das 2-Zustands-Prozessmodell

- Prinzipiell genügen 2 Prozesszustände
  - **rechnend**: Einem Prozess wurde die CPU zugeteilt
  - **nicht rechnend**: Der Prozess wartet auf die Zuteilung der CPU
- Da die nicht-Zuteilung der CPU für einen Prozess verschiedene Ursachen haben kann, kann das Prozessmodell praktisch beliebig verfeinert werden



2-Zustands-Prozessmodell (Implementierung)

- Die Prozesse im Zustand **nicht rechnend** müssen in einer Warteschlange gespeichert werden, in der sie auf ihre Ausführung warten



- Dieses Modell beschreibt auch das Verhalten des **Dispatchers**
  - Aufgabe des Dispatchers ist die Umsetzung der Zustandsübergänge
- Die Bestimmung der Reihenfolge, welcher Prozess als nächstes Zugriff auf die CPU erhält, regelt das verwendete **Scheduling**-Verfahren

Konzeptioneller Fehler des 2-Zustands-Prozessmodells

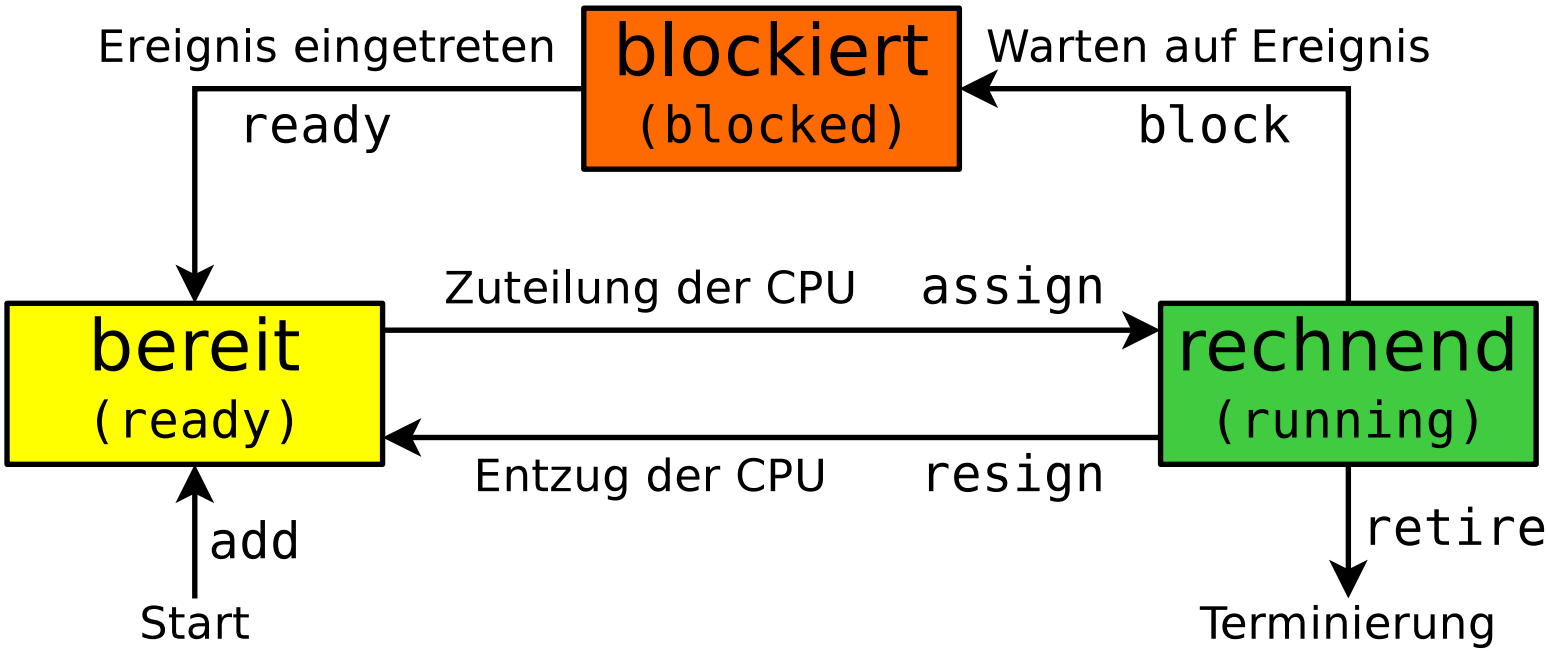
- Das 2-Zustands-Prozessmodell geht davon aus, dass alle Prozesse immer zur Ausführung bereit sind
  - Das ist unrealistisch!
- Es gibt fast immer Prozesse, die **blockiert** sind
- Mögliche Gründe, warum Prozesse blockiert sind:
  - Warten auf die Eingabe oder Ausgabe eines E/A-Geräts
  - Warten auf das Ergebnis eines anderen Prozesses
  - Warten auf eine Reaktion des Benutzers
- Lösung: Die nicht rechnenden Prozesse müssen in zwei Gruppen unterschieden werden
  - Prozesse die **bereit** (ready) sind
  - Prozesse die **blockiert** (blocked) sind

$\Longrightarrow$ 3-Zustands-Prozessmodell

3-Zustands-Prozessmodell

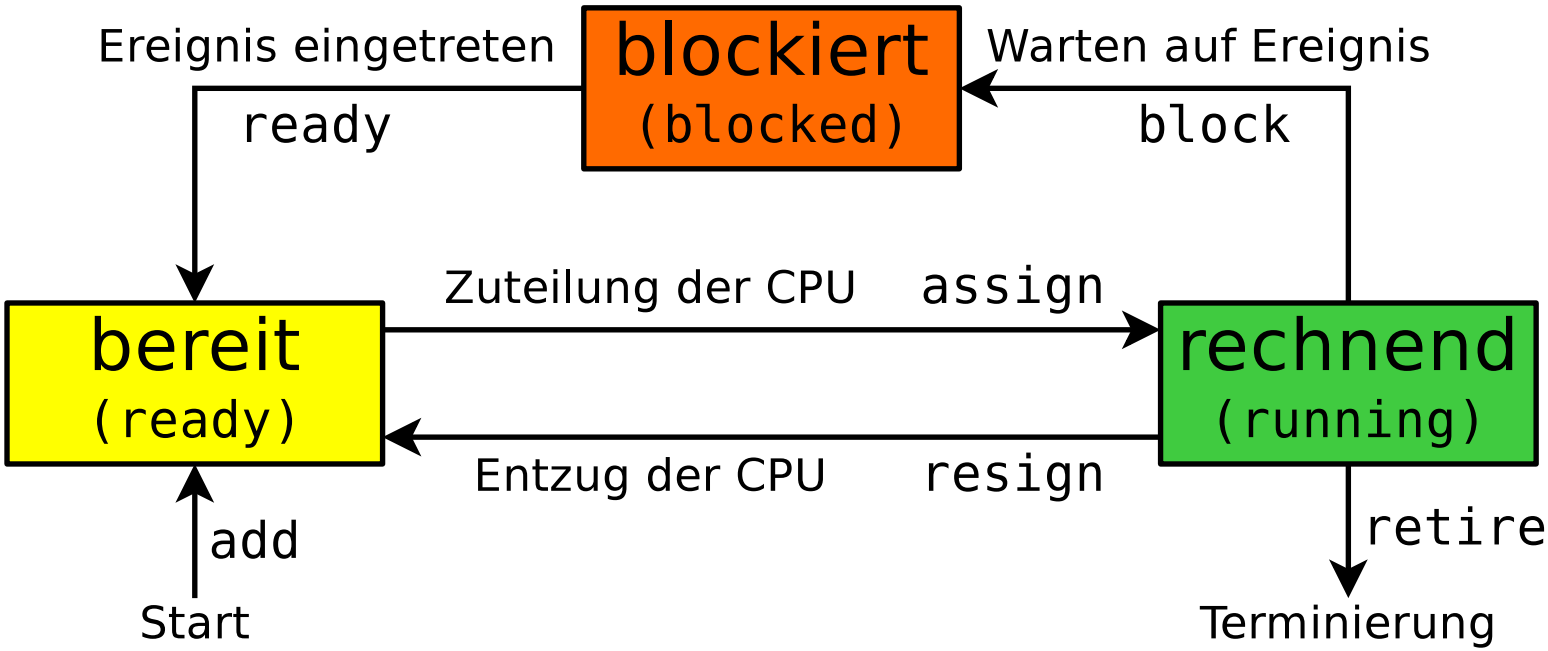
- Jeder Prozess befindet sich in einem der folgenden 3 Zustände
- **rechnend** (*running*):
  - Der Prozess besitzt den Zugriff auf die CPU und führt auf dieser Instruktionen aus
- **bereit** (*ready*):
  - Der Prozess könnte unmittelbar Instruktionen auf der CPU ausführen und wartet aktuell auf die Zuteilung der CPU
- **blockiert** (*blocked*):
  - Der Prozess kann momentan nicht weiter ausgeführt werden und wartet aktuell auf das Eintreten eines Ereignisses oder einer Bedingung
  - Dabei kann es sich z.B. um eine Nachricht eines anderen Prozesses oder eines Eingabe-/Ausgabegeräts oder aber um das Eintreten eines bestimmten Synchronisationsereignisses handeln

Prinzip des 3-Zustands-Prozessmodells (1)



- **add**: Prozesserzeugung durch einen Programmstart oder einen anderen Prozess und Einordnung in die Liste der Prozesse im Zustand `bereit`
- **retire**: Der aktuell rechnende Prozess terminiert
  - Alle durch den Prozess belegten Ressourcen werden freigegeben
- **assign**: Die CPU wird einem Prozess im Zustand `bereit` zugeteilt, der nun mit der CPU arbeiten kann und in den Zustand `rechnend` wechselt

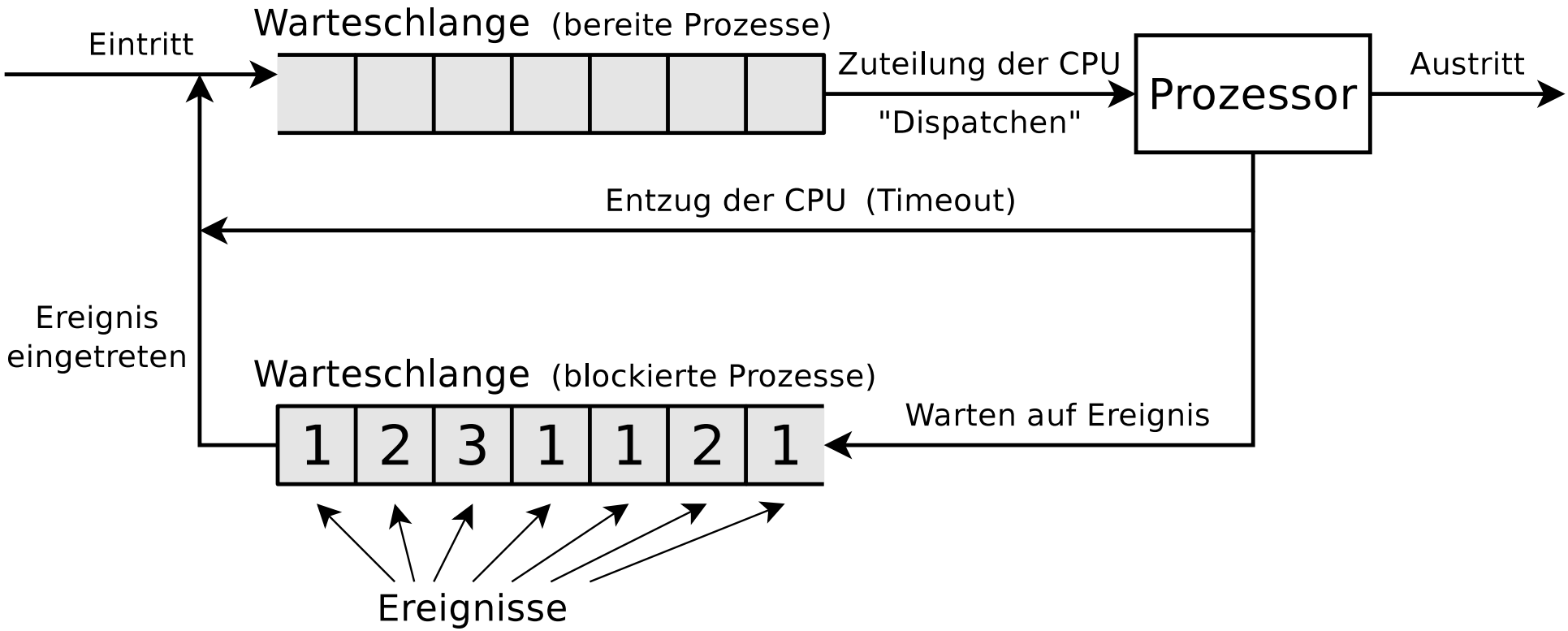
Prinzip des 3-Zustands-Prozessmodells (2)



- **block**: Der rechnende Prozess wartet auf eine Nachricht oder ein Synchronisationsereignis und wechselt in den Zustand `blockiert`
- **resign**: Dem rechnenden Prozess wird wegen einer Entscheidung des Schedulers die CPU entzogen und er wechselt in den Zustand `bereit`
- **ready**: Die Bedingung, wegen der der Prozess blockiert wurde, ist nun erfüllt und der Prozess wechselt in den Zustand `bereit`

Das 3-Zustands-Prozessmodell – Implementierung (1/2)

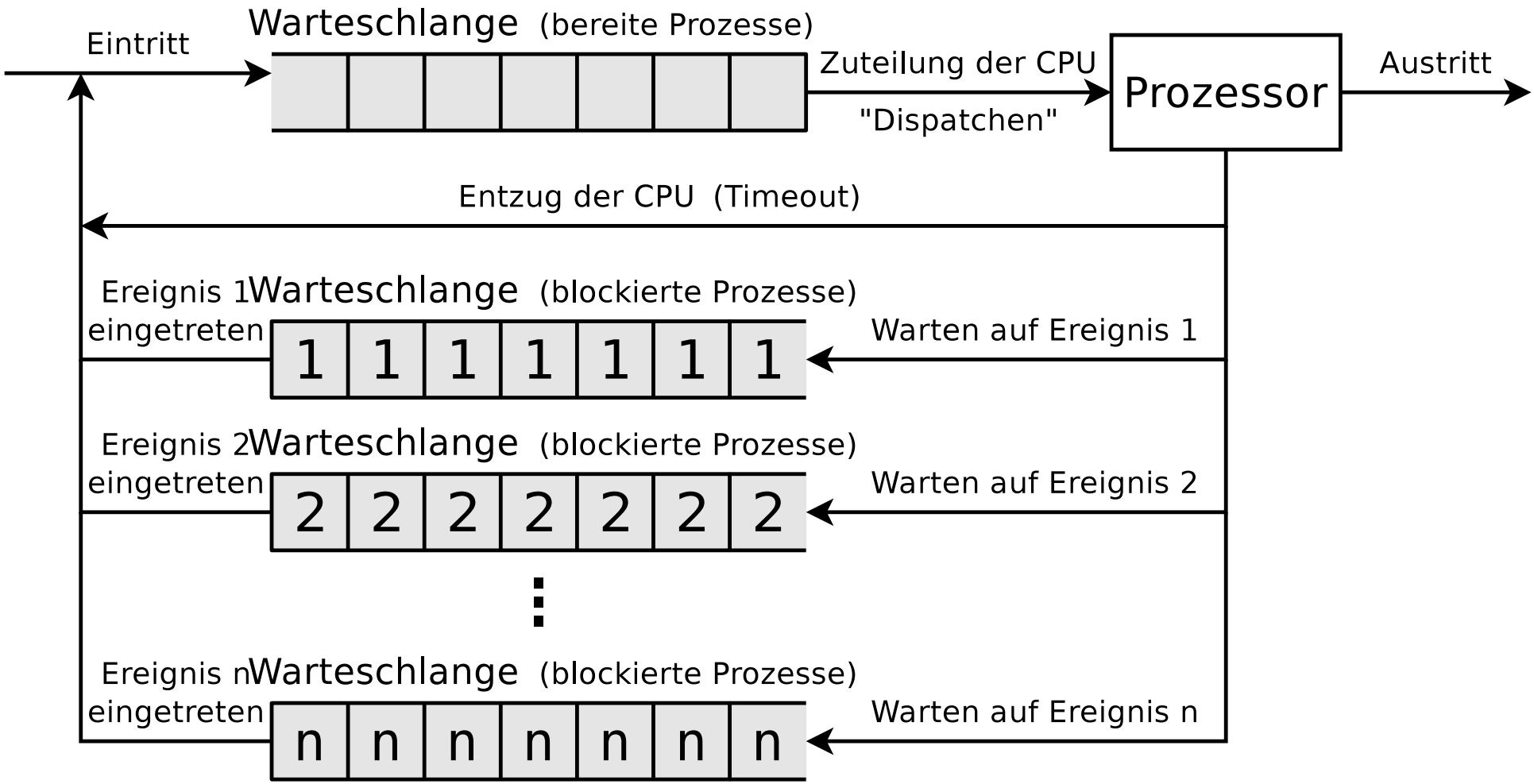
- Eine Implementierung könnte auf 2 Warteschlangen basieren
  - Warteschlange für die Prozesse im Zustand `bereit`
  - Warteschlange für die blockierten Prozesse



- Mehreren Warteschlangen für die blockierten Prozesse sind sinnvoll

Das 3-Zustands-Prozessmodell – Implementierung (2/2)

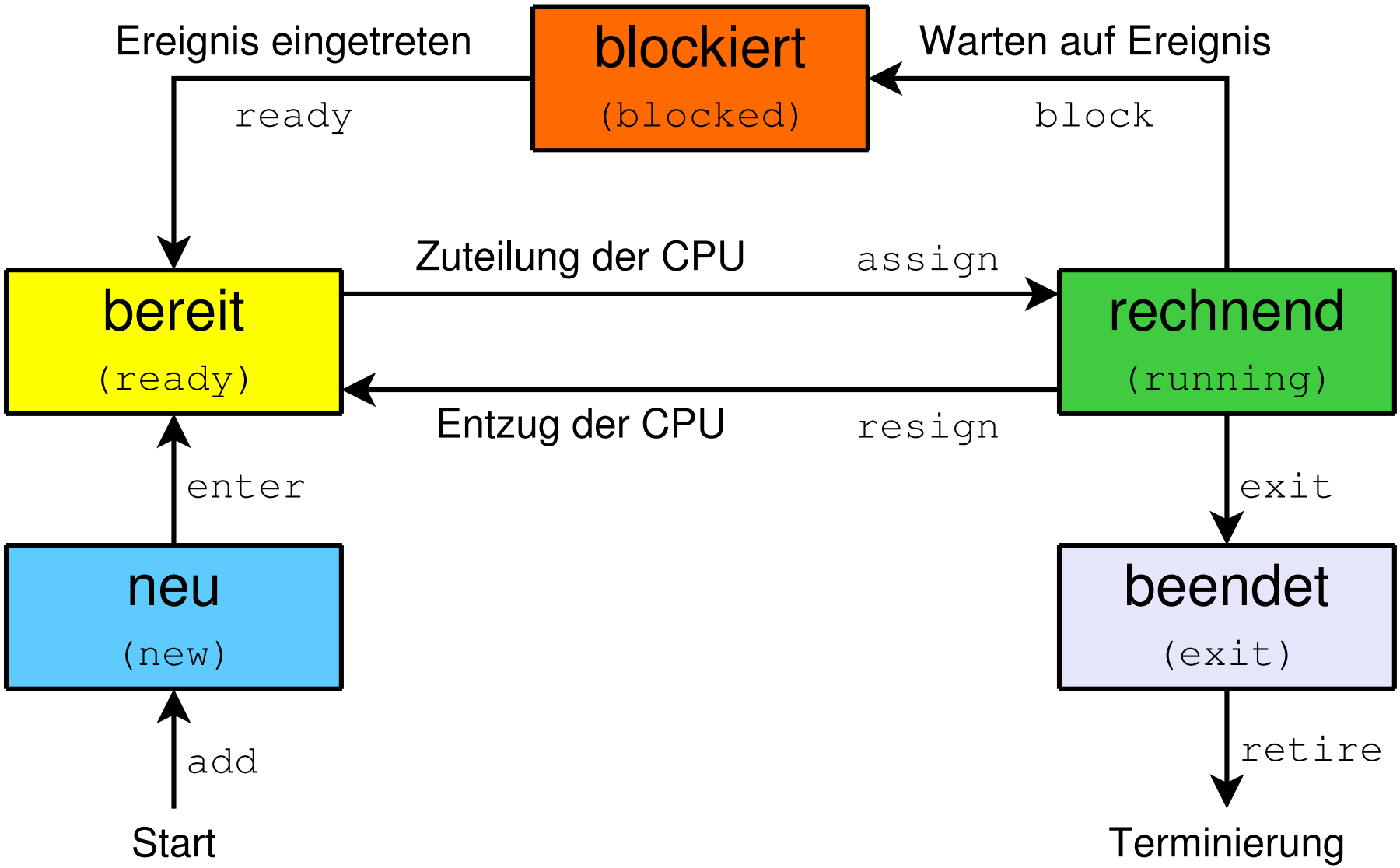
- Mehrere Warteschlangen für die blockierten Prozesse



Das 5-Zustands-Prozessmodell

- Es kann empfehlenswert sein, das 3-Zustands-Prozessmodell um 2 weitere Prozesszustände zu erweitern
  - **neu** (*new*): Der Prozess (Prozesskontrollblock) ist erzeugt, wurde aber vom Betriebssystem noch nicht der Menge (Warteschlange) der Prozesse zugefügt, die unmittelbar Instruktionen auf der CPU ausführen könnten und auf die Zuteilung der CPU warten
  - **exit** (*exit*): Der Prozess (Prozesskontrollblock) existiert noch, wurde vom Betriebssystem aber aus der Menge (Warteschlange) der ausführbaren Prozesse entfernt
- Grund für die Existenz der Prozesszustände **neu** und **exit**:
  - Auf manchen Systemen ist die Anzahl der ausführbaren Prozesse limitiert, um Ressourcen (Speicher) zu sparen und den Grad des Mehrprogrammbetriebs festzulegen

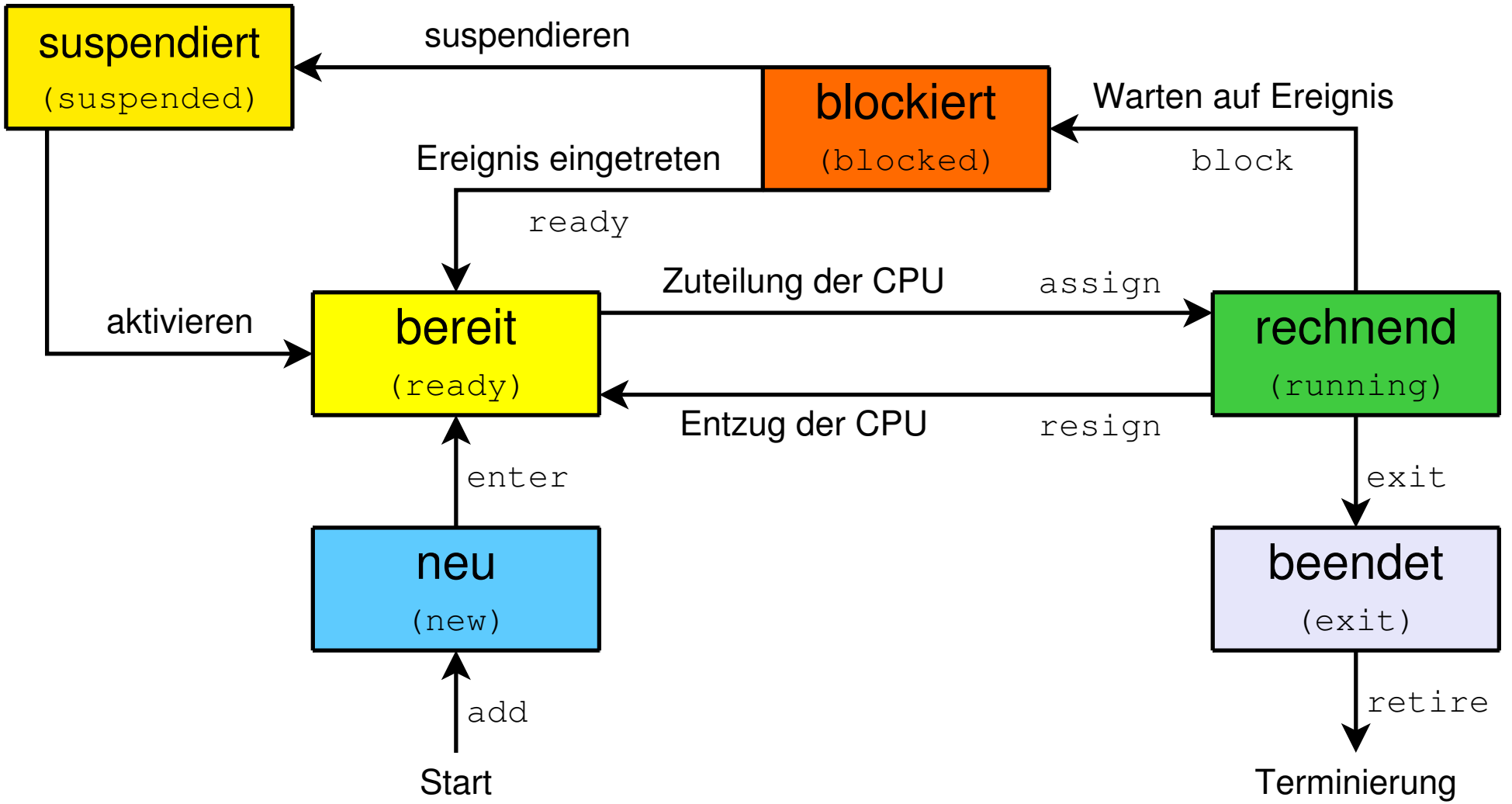
Prinzip des 5-Zustands-Prozessmodells



Das 6-Zustands-Prozessmodell

- Ist nicht genügend physischer Hauptspeicher für alle laufenden Prozesse verfügbar, was häufig der Fall ist, müssen Teile von Prozessen oder ganze Prozesse ausgelagert werden
- **Swapping** = Vorgang des Auslagerns
- Das Betriebssystem lagert Prozesse aus, die im Zustand **blockiert** sind
  - Ausgelagerte Prozesse werden in eine Warteschlange auf die Festplatte geschrieben
- Durch freigewordenen Platz können Prozesse in der Warteschlange der ausgelagerten Prozesse oder neue Prozess in den Speicher geladen und von der CPU ausgeführt werden
  - Es macht also Sinn, das 5-Zustands-Prozessomodell um den Prozesszustand **suspendiert** (*suspended*) zu erweitern

Prinzip des 6-Zustands-Prozessmodells

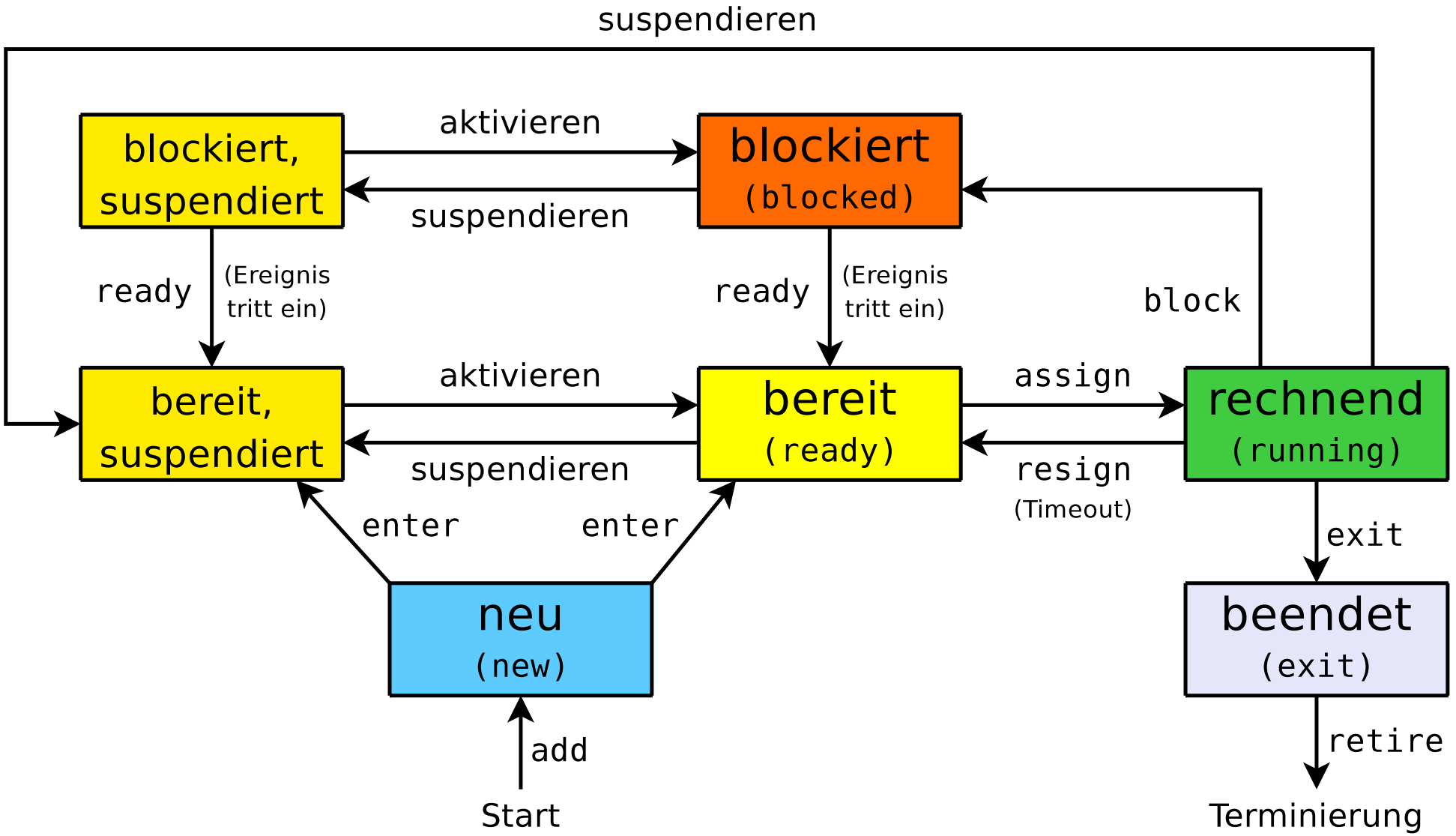


Optimierung des 6-Zustands-Prozessmodells

- Wurde ein Prozess ausgelagert (suspendiert), ist es besser, den frei gewordenen Platz im Hauptspeicher zu verwenden, um einen ausgelagerten Prozess zu aktivieren, als einen neuen Prozess zu nehmen
  - Dieses Vorgehen macht aber nur Sinn, wenn der aktivierte Prozess nicht mehr blockiert ist
- Was im 6-Zustands-Prozessmodell fehlt, ist die Möglichkeit, die ausgelagerten Prozesse in blockierte und nicht-blockierte Prozesse zu unterscheiden

$\Longrightarrow$ 7-Zustands-Prozessmodell

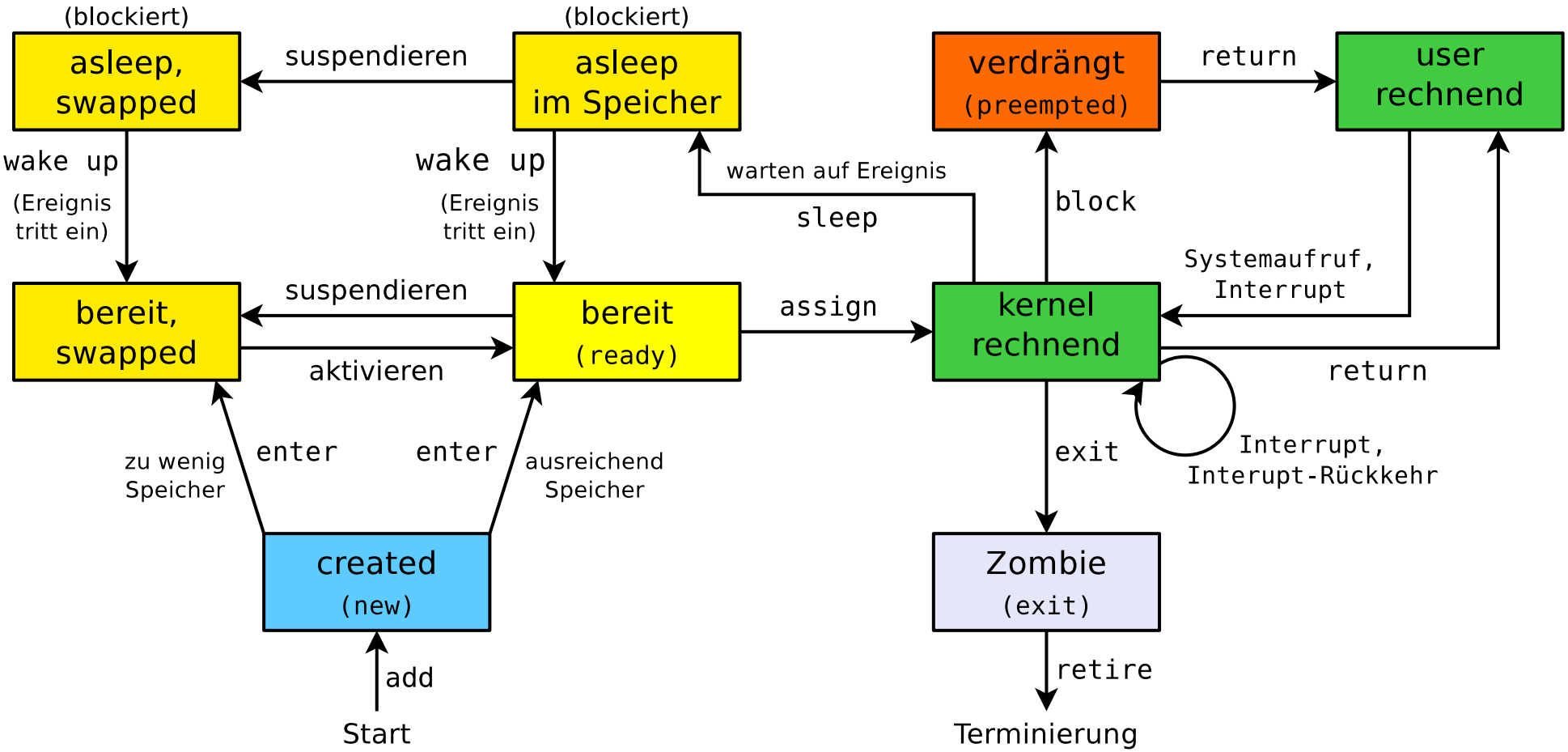
Prinzip des 7-Zustands-Prozessmodells



Prozessmodell von Linux/UNIX

- Unter Linux/UNIX-Betriebssystemen gilt das 9-Zustands-Prozessmodell
- Der Zustand **rechnend** (*running*) wird unterteilt in die Zustände
  - **benutzer rechnend** (*user running*) für Prozesse im Benutzermodus
  - **kernel rechnend** (*kernel running*) für Prozesse im Kernel-Modus
- Der Zustand **bereit** (*ready*) wird unterteilt in die Zustände
  - **bereit** (*ready*) für Prozesse, die neu in den Speicher geladen oder aufgeweckt wurden
  - **verdrängt** (*preempted*) für Prozesse, die durch einen Timeout oder eine Vorrangunterbrechung in ihrer Ausführung unterbrochen wurden
    - Eine Verdrängung kann nur erfolgen, wenn ein Prozess gerade vom Kernel-Modus in den Benutzermodus wechselt
    - Ist ein Prozess im Kernel-Modus, kann er nicht verdrängt werden
- Einige Prozesszustände haben unter Linux/UNIX andere Namen
  - Der Prozesszustand **exit** heißt **zombie**
  - Der Prozesszustand **suspendiert** heißt **swapped**
  - Der Prozesszustand **blockiert** heißt **asleep**
  - Der Prozesszustand **neu** heißt **created**

Prinzip des 9-Zustands-Prozessmodells



Prozesse

- Das Betriebssystem führt Buch über die laufenden Prozesse
  - Es muss wissen, welche Prozesse aktuell bearbeitet werden müssen
  - Es muss wissen, welche Eigenschaften die Prozesse haben
- Wird ein Prozess angehalten, muss er zu einem späteren Zeitpunkt in dem Zustand, in dem er gestoppt wurde, wieder gestartet werden
  - Alle wichtigen Informationen über den Prozess müssen für die Dauer der Unterbrechung zwischengespeichert werden
  - Hat ein Prozess zum Zeitpunkt der Unterbrechung Dateien geöffnet, muss für jede Datei der Zeiger der aktuellen Position innerhalb der Datei gespeichert werden
- In der **Prozesstabelle** speichert das Betriebssystem alle Informationen zu allen Prozessen des Systems gesammelt
  - Einzige Ausnahme: Der Inhalt des Adressraums wird nicht in der Prozesstabelle gespeichert
- Die Prozesstabelle wird als Array oder verkettete Liste realisiert
  - Für jeden Prozess existiert eine eigene Tabelle, der **Prozesskontrollblock**

Prozesstabelle und Prozesskontrollblöcke (1)

- Jeder Prozess hat eine **Prozessidentifikation** (*process identifier*), kurz PID genannt, zur eindeutigen Identifikation

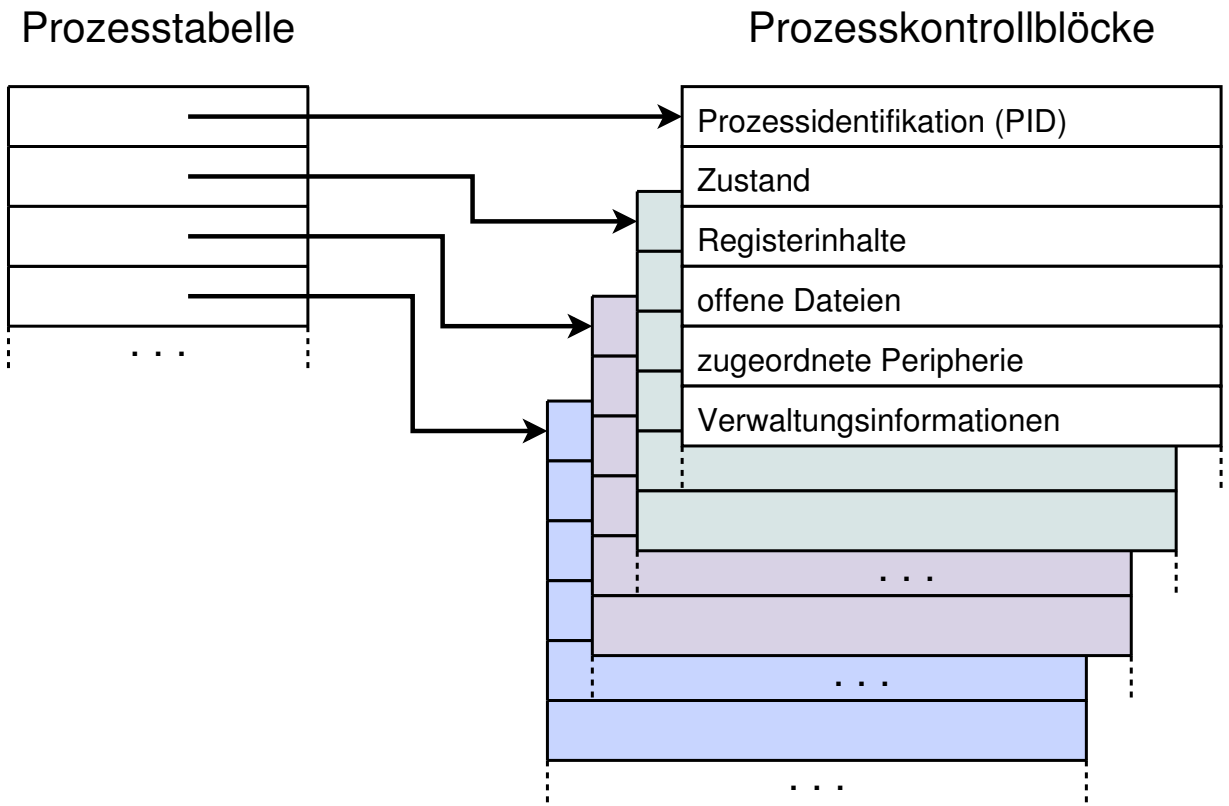
```
baun@olymp:~$ ps
  PID TTY          TIME CMD
 1876 pts/3    00:00:04 bash
 2411 pts/3    00:00:00 ps
```

- Im Prozesskontrollblock jedes Prozesses befindet sich:
  - Die Prozessidentifikation (PID)
  - Der aktuelle Prozesszustand (`rechnend`, `blockiert`, `bereit`)
  - Inhalt der Prozessorregister (Befehlszähler, Stack Pointer, usw.)
  - Eine Liste mit zugeordneten Bereichen im Hauptspeicher
  - Vom Prozess geöffnete Daten und zugeordnete Peripheriegeräte
  - Verwaltungs-/Schedulinginformationen (Priorität, Laufzeit, usw.)

TTY = Teletypewriter, PTS = Pseudo Terminal

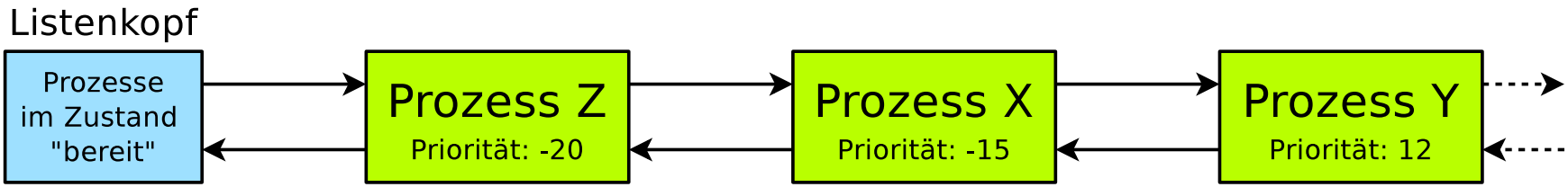
Prozesstabelle und Prozesskontrollblöcke (2)

- Der Prozesskontrollblock enthält die Informationen, die nötig sind, wenn ein Prozess vom Zustand `rechnend` in die Zustände `bereit` oder `blockiert` übergeht, um nach der Unterbrechung ein nahtloses Weiterlaufen des Prozesses zu garantieren



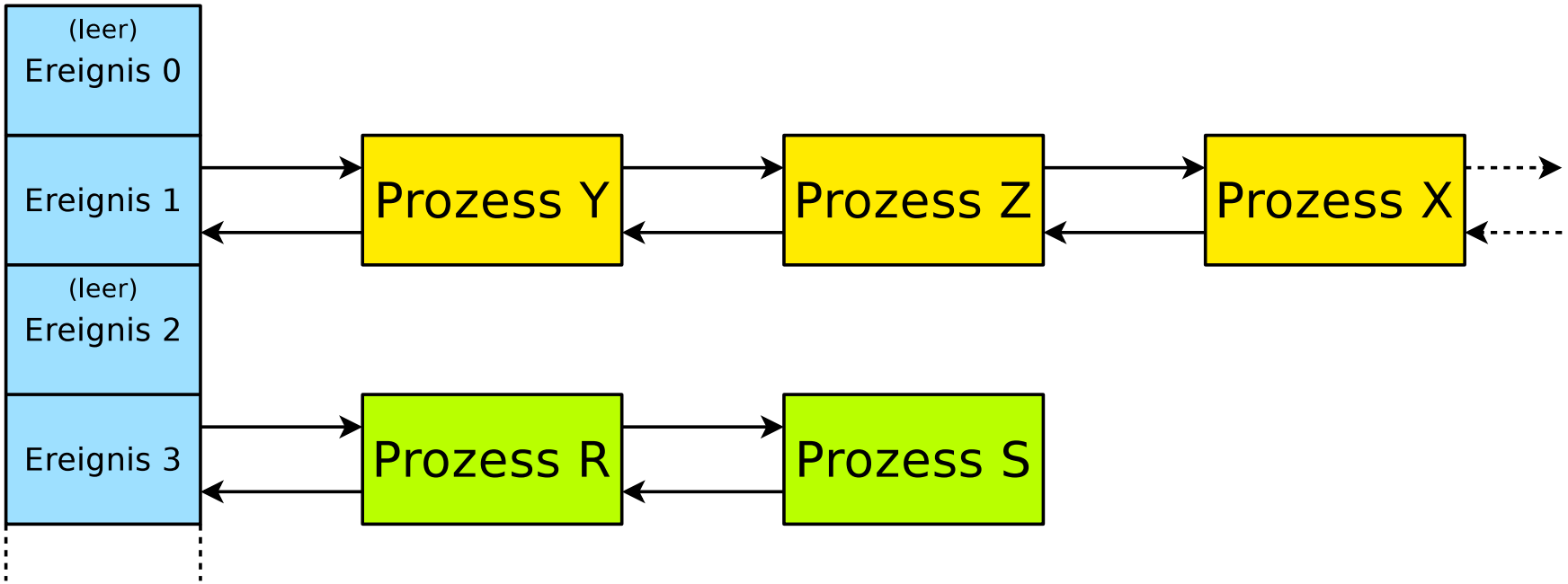
Zustandslisten – `bereit`-Liste

- Neben der Prozesstabelle mit den Prozesskontrollblöcken führt das Betriebssystem verkettete Listen für die Prozesse mit den Zuständen `bereit` und `blockiert`
- Die Liste der Prozesse mit dem Zustand `bereit` enthält alle Prozesse, die unmittelbar ausgeführt werden können, aber auf die Zuteilung des Prozessors warten
- Die Prozesse in der `bereit`-Liste können nach unterschiedlichen Kriterien, u.a. nach den Prozessprioritäten oder der Wartezeit, sortiert werden

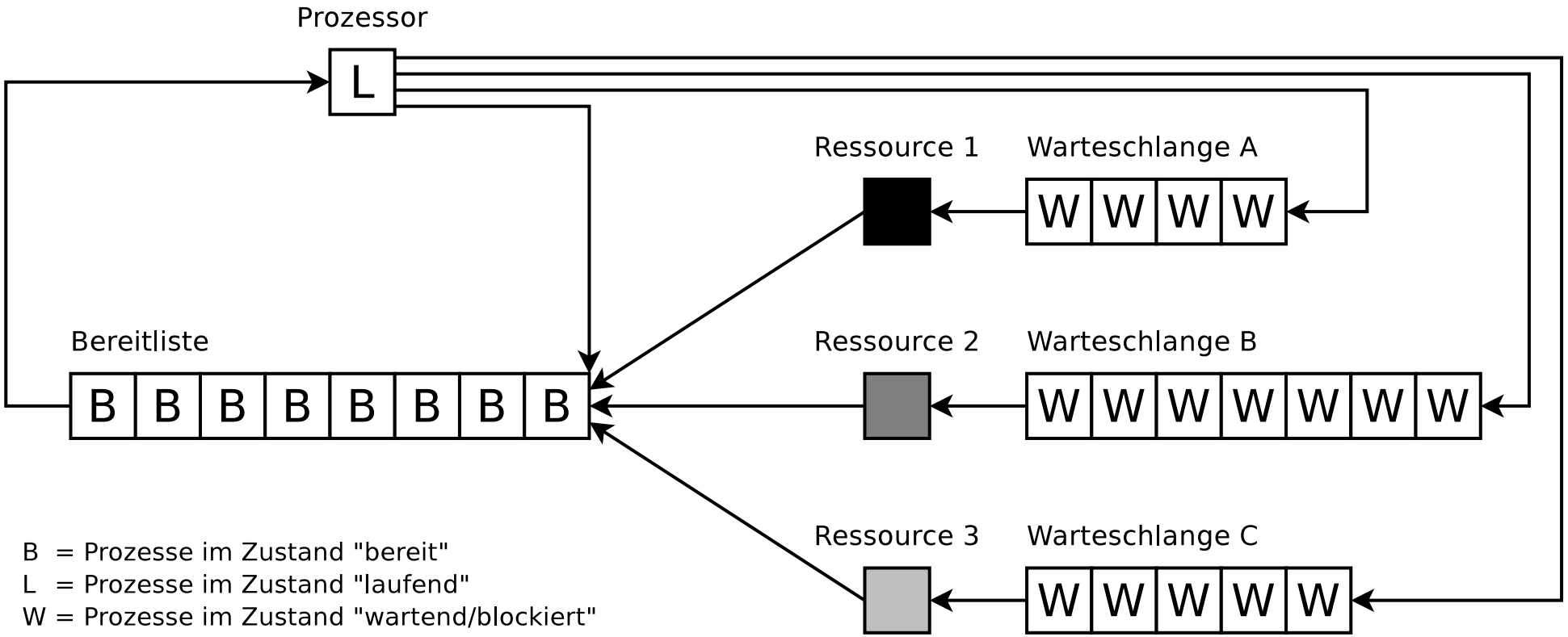


Zustandslisten – `blockiert`-Liste

- Für die Prozesse im Zustand `blockiert` gibt es mehrere Listen
- Jedem Ereignis, auf das einer oder mehr Prozesse warten, ist eine eigene Liste zugeordnet, in der alle Prozesse aufgeführt sind, die auf das Ereignis warten



Konzeptionelle Prozessverwaltung



Quelle: Eduard Glatz, Betriebssysteme, dpunkt Verlag, 2010

Zustandsübergänge

- Für einen Zustandsübergang eines Prozesses wird der Prozesskontrollblock des betreffenden Prozesses aus der alten Zustandsliste entfernt und in die neue Zustandsliste eingefügt
- Für die Prozesse mit dem Zustand `rechnend` gibt es keine eigene Liste
  - Es gibt aber eine Variable des Betriebssystems, die auf den Prozesskontrollblock des aktuell rechnenden Prozesses zeigt
- Beim Übergang eines Prozesses in den Zustand `rechnend` werden die Registerinhalte des Prozessors aus dem Prozesskontrollblock geladen
- Beim Übergang aus dem Zustand `rechnend` in einen anderen Zustand werden die Registerinhalte des Prozessors in den Prozesskontrollblock gesichert (gerettet)

Nächste Vorlesung

Nächste Vorlesung:

1.12.2011